BlackBerry Java SDK

Smart Card

Version: 7.1

Development Guide

BlackBerry

Contents

Drivers for smart cards and smart card readers

1

BlackBerry devices include the drivers that support the use of certain smart cards. The supported smart card readers require associated drivers. If you want to use a supported smart card on a supported reader, you do not have to write a driver for the smart card or a driver for the smart card reader. In the other potential scenarios of smart card and smart card reader usage you must create and register the drivers.

Scenario	Implementation requirements
supported reader and unsupported card	Create and register a driver for the smart card by implementing the abstract `SmartCard` or `CryptoSmartCard` class and the abstract `SmartCardSession` class.
unsupported reader and supported card	Create and register a driver for the smart card reader by implementing the abstract `SmartCardReader` class and the abstract `SmartCardReaderSession` class.
unsupported reader and unsupported card	Create and register a driver for the smart card by implementing the abstract `SmartCard` or `CryptoSmartCard` class and the abstract `SmartCardSession` class. Also, create and register a driver for the smart card reader by implementing the abstract `SmartCardReader` class and the abstract `SmartCardReaderSession` class.

To create drivers that interact with smart cards and the BlackBerry Smart Card Readeryou can use the `net.rim.device.api.smartcard` package and the `net.rim.device.api.crypto` packages in the Smart Card API that is provided in BlackBerry Java Development Environment 4.1.

A driver for a cryptographic smart card that implements the Smart Card API can work with the S/MIME Support Package for BlackBerry smartphones on a BlackBerry device with S/MIME support. A driver for a cryptographic smart card can perform signing private key operations on the smart card such as signing and decrypting messages. A driver for a cryptographic smart card does not require the S/MIME Support Packagefor BlackBerry smartphones to be able to import certificates from the smart card, or to provide two-factor authentication for a BlackBerry device. For more information on S/MIME, see the *BlackBerry with the S/MIME Support Package*.

The Smart Card API that is provided in BlackBerry JDE 4.2 or later contains some deprecated elements. The deprecated elements provide backward compatibility if you create a driver for a cryptographic smart card for BlackBerry devices with BlackBerry Device Software 4.1.x. If you want to create a driver for a cryptographic smart card for BlackBerry devices with BlackBerry Device Software version 4.1.x or version 4.2 or later, you can use the deprecated elements to avoid creating two versions of the driver.

If you want to create a driver for a cryptographic smart card for BlackBerry devices with BlackBerry Device Software 4.2 or later, use the nondeprecated API elements in the Smart Card API.

Supported smart cards

2

BlackBerry devices include drivers that enable the use of the smart cards listed in the following table. You can create drivers to support additional smart cards by implementing the abstract `SmartCard` and `SmartCardSession` classes.

Smart card	Description
CAC	CAC smart cards are the cards that are used by the United States Department of Defense.
PIV	PIV smart cards are the cards that are used by United States government employees and contractors.
SafeNet Model 330	The SafeNet Model 330 smart card is a cryptographic smart card with an embedded microcontroller and 32K of memory.

Smart card driver infrastructure 3

BlackBerry devices include a software infrastructure that enables them to communicate with a smart card by interacting with the smart card reader the card is inserted in.

When you insert a smart card into a smart card reader, the reader driver retrieves the ATR of the smart card. As a reader driver developer, you must provide an implementation for `SmartCardReaderSession.getAnswerToResetImpl()` that retrieves and returns the ATR of the smart cards the reader supports.

As a smart card driver developer, you must provide an implementation of `SmartCard.checkAnswerToResetImpl(AnswerToReset atr)`, that returns `True` if your driver should be used to communicate with the smart card with the provided ATR.

Creating a cryptographic smart card driver 4

To create a cryptographic smart card driver for BlackBerry Device Software version 4.1 or later, you have to extend two classes. One class, `CryptoSmartCard`, represents a type of smart card. The other class, `CryptoSmartCardSession`, represents a communications session with a smart card of that type.

Create a smart card driver project

Smart card drivers are library applications. Follow these steps to create a library application project.

1. On the **File** menu, click **New > Project**.
2. Expand the BlackBerry® folder and select **BlackBerry Project**.
3. Click **Next**.
4. In the **Project name** field, type a name for the project.
5. Select **Create new project in workspace**.
6. Complete one of the following tasks:
   - To specify a specific JRE, select **Use a project specific JRE**.
   - To specify the default JRE in the workspace, select **Use default JRE**.
7. Click **Next**.
8. Click **Finish**.
9. In the **Package Explorer** view, right-click a BlackBerry application project and click **Properties**.
10. In the **Properties for** pane, click **BlackBerry Project**.
11. Click **Application Descriptor**.
12. Click the **Application** tab.
13. In the **Application type** field, select **Library**.
14. Check the **Auto-run on startup** check box.
15. In the **Startup tier field**, select **7**.
16. On the Eclipse toolbar, click **Save**.

CryptoSmartCard methods you implement to create a driver

Methods you must implement for a smart card driver

Method to implement	Description
SmartCard.openSessionImpl(SmartCardReaderSession)	Opens a cryptographic session with a smart card.
SmartCard.checkAnswerToResetImpl(AnswerToReset)	Verifies if a smart card is compatible with a specific Answer To Reset (ATR) sequence.
SmartCard.getCapabilitiesImpl()	Returns an instance of the SmartCardCapabilities class that enumerates the capabilities of the smart card.

Method to implement	Description
SmartCard.displaySettingsImpl(Object)	Enables a smart card driver to display settings or properties.
SmartCard.isDisplaySettingsAvailableImpl(Object)	Enables a smart card driver to indicate support for display settings.
SmartCard.getCapabilitiesImpl()	Retrieves the capabilities of a smart card. The capabilites of a smart card include the protocols the card supports, the baud rate, and the clock adjustment factors.
SmartCard.getLabelImpl()	Retrieves the smart card type.
CryptoSmartCard.getAlgorithms()	Retrieves the names of the algorithms that the smart card supports, for example (“RSA”, “DSA”).
CryptoSmartCard.getCryptoToken(String)	Retrieves a `CryptoToken` object that supports the given algorithm.
SmartCard.getLabelImpl()	Returns a label associated with the kind of smart card.

Implement a class to represent a type of smart card

1. Import the required crypto and smartcard libraries.
2. Extend the `CryptoSmartCard` class.
3. Implement a `libMain()` method that calls `SmartCardFactory.addSmartCard()`
4. Implement the required `CryptoSmartCard` methods.

CryptoSmartCardSession methods you implement to create a driver

Methods that you must implement for any smart card driver.

CryptoSmartCardSession methods	Description
`SmartCardSession.closeImpl()`	Closes a cryptographic smart card session.
SmartCardSession.getMaxLoginAttemptsImpl()	Retrieves the maximum number of login attempts.
SmartCardSession.getRemainingLoginAttemptsImpl()	Retrieves the remaining number of login attempts.
SmartCardSession.getSmartCardIDImpl()	Retrieves the ID for the smart card.
SmartCardSession.loginImpl(String)	Attempts to log in to the cryptographic session using a given password string.
CryptoSmartCardSession.getKeyStoreDataArrayImpl()	Retrieves certificates from the smart card.
CryptoSmartCardSession.getRandomBytesImpl(int maxNumBytes)	Retrieves random data from the internal random number generator of the smart card.

Additional methods that you must implement for smart card drivers that support retrieving fingerprint data from smart cards.

Method to implement	Description
CryptoSmartCardSession.getFingerprintsImpl(int context)	Extend this method to provide support for extracting fingerprint sample data from a smart card. The data must be returned in a `FingerprintBiometricData[]`. The context parameter indicates why the fingerprints are being requested. In BlackBerry Java Development Environment 5.0, only `FINGERPRINT_CONTEXT_AUTHENTICATION` is used.

Implement a class to enable a communication session with a smart card

1. Import the required crypto and smartcard libraries.
2. Extend the abstract `CryptoSmartCardSession` class.
3. Implement `CryptoSmartCardSession.getKeyStoreDataArrayImpl()` according to the BlackBerry Device Software version of the target device.
   - To create a cryptographic smart card driver that is compatible with BlackBerry Device Software version 4.2 or later, implement `CryptoSmartCardSession.getKeyStoreDataArrayImpl()` as follows:

     ```
     RSACryptoToken token = new MyRSACryptoToken();
     RSACryptoSystem cryptoSystem = new RSACryptoSystem(token, 1024);
     RSAPrivateKey privateKey;
     PrivateKey privateKey = new RSAPrivateKey(cryptoSystem, new
     MyCryptoTokenData());
     ```

   - To create a cryptographic smart card driver that is compatible with BlackBerry Device Software version 4.1 and version 4.2 or later, and to include the cryptographic smart card driver in two-factor authentication, implement the `getKeyStoreDataArrayImp` method as follows:

     ```
     PrivateKey privateKey = CryptoSmartCardUtilities2.createPrivateKey(token, 1024,
     new MyCryptoTokenData());
     ```

4. Implement `CryptoSmartCardSession.getKeyStoreDataArrayImpl()`
5. Implement the other required `CryptoSmartCardSession` methods.

Verify installation of a cryptographic smart card driver

5

1. On the BlackBerry device, click **Options > Security Options > Smart Card**.
2. Ensure the cryptographic smart card appears in the **Registered Card Drivers** section.

Code samples

6

Code sample: List smart card drivers

```
import net.rim.device.api.smartcard.SmartCardFactory;
import net.rim.device.api.smartcard.SmartCard;
import net.rim.device.api.ui.UiApplication;
import net.rim.device.api.ui.component.LabelField;
import net.rim.device.api.ui.container.MainScreen;

public class ListSmartCards extends UiApplication
{
    public static void main(String args[])
    {
        ListSmartCards app = new ListSmartCards();
        app.enterEventDispatcher();
    }

    public ListSmartCards()
    {
        HomeScreen hs = new HomeScreen();
        pushScreen(hs);
    }
}

class HomeScreen extends MainScreen
{
    public HomeScreen()
    {
        int numCards = SmartCardFactory.getNumSmartCards();
        SmartCard[] smartCards = SmartCardFactory.getSmartCards();
        int len = smartCards.length;
        StringBuffer sbCards = new StringBuffer();
        for(int i=0;i<len;i++)
        {
            sbCards.append(smartCards[i].getLabel() + '\n');
        }
        LabelField msg = new LabelField("There are " + Integer.toString(numCards)
        + " smart card drivers installed on this device:\n");
        LabelField msgCards = new LabelField(sbCards.toString());
        add(msg);
        add(msgCards);
     }
}
```

Code sample: Sign data using private key on smart card and verify on the device

```
import java.util.Enumeration;
import net.rim.device.api.crypto.Crypto;
```

```
import net.rim.device.api.crypto.CryptoException;
import net.rim.device.api.crypto.NoSuchAlgorithmException;
import net.rim.device.api.crypto.PrivateKey;
import net.rim.device.api.crypto.RandomSource;
import net.rim.device.api.crypto.certificate.*;
import net.rim.device.api.crypto.keystore.*;
import net.rim.device.api.system.Application;

public class SignDataPrivate extends Application
{
  public static void main(String[] args)
  {
    SignDataPrivate app = new SignDataPrivate();
    KeyStore keyStore;
    try
    {
      keyStore = DeviceKeyStore.getInstance();
      KeyStoreData ksData = null;
      AssociatedDataKeyStoreIndex index = new AssociatedDataKeyStoreIndex(
      AssociatedData.SMART_CARD_KEY);
                        keyStore.addIndex(index);
      Enumeration enumeration = keyStore.elements( index.getID() );
      while( enumeration.hasMoreElements() )
      {
        ksData = (KeyStoreData) enumeration.nextElement();
        Certificate certificate = ksData.getCertificate();
        if( ( certificate == null ) || !ksData.isPrivateKeySet())
        {
          continue;
        }
        if( !( certificate.queryKeyUsage(
               KeyUsage.DIGITAL_SIGNATURE ) != KeyUsageResult.NOT_ALLOWED ) )
        {
          continue;
        }
      }

      byte [] randomData = new byte[ 64 ];
      RandomSource.getBytes( randomData );
      PrivateKey privateKey = ksData.getPrivateKey( null );
      byte [] signature = Crypto.sign(randomData,0,randomData.length,
                                      privateKey,null,"X509" );

      if(!Crypto.verify(randomData, 0, randomData.length,
                        keyStoreData.getPublicKey(), "X509", signature, 0 ))
      {
         return false;
      }
    }
    catch (KeyStoreRegisterException e)
    {
    }
    catch( NoSuchAlgorithmException e )
    {
    }
    catch( CryptoException e )
```

```
    {
    }
    catch( IllegalArgumentException e )
    {
    }
  }
}
```

Code sample: Implementing a class to enable a communication session with a smart card

```
package com.rim.samples.device.smartcard;
import net.rim.device.api.crypto.*;
import net.rim.device.api.crypto.certificate.*;
import net.rim.device.api.crypto.certificate.x509.*;
import net.rim.device.api.crypto.keystore.*;
import net.rim.device.api.smartcard.*;
import net.rim.device.api.util.*;

/**************************************************************************
* This class represents a communication session with a physical smart card.
*
* Application Protocol Data Units (APDUs) can be sent to the smart card over a
* session to provide the desired functionality.
*
* Do not keep sessions open when you are not using them. They should be
* short-lived. As a security precaution, only one open session is allowed
* per SmartCardReader.Subsequent openSession() requests will block until the
* current session is closed.
*/

     public class MyCryptoSmartCardSession extends CryptoSmartCardSession
     {

        // We assume that the smart card has three certificates identified
        // by: ID_PKI, SIGNING_PKI and ENCRYPION_PKI. Your smart card may have a
        // different number of certificates or be identified differently. These
        // three certificates are just an example of what might be stored on a
        // smart card.

     public static final byte ID_PKI             = (byte)0x00;
     public static final byte SIGNING_PKI        = (byte)0x01;
     public static final byte ENCRYPTION_PKI     = (byte)0x02;
     private static final String WAITING_MSG     = "Please Wait";
     private static final String ID_STRING       = "Jason Hood";
     private static final String ID_CERT         = "ID Certificate";
     private static final String SIGNING_CERT    = "Signing Certificate";
     private static final String ENCRYPTION_CERT = "Encryption Certificate";


     /**************************************************************************
```

```
     * Provide a constructor for the session.
     *
     * The smartCard argument is the smart card associated with this session.
     * The readerSession argument holds the reader session commands sent to this
     * smart card.
     ********************************************************************************/

     protected MyCryptoSmartCardSession(SmartCard smartCard, SmartCardReaderSession
readerSession)
     {
          super( smartCard, readerSession );
     }

     /********************************************************************************
     * Provide a method to close the session.
     *
     * Do not close the underlying SmartCardReaderSession.Use this method to
     * clean up the session prior to its closure.
     ********************************************************************************
     /
     protected void closeImpl()
     {
        // Do any session cleanup needed here.
     }

     /********************************************************************************
     * Provide a method that returns the maximum number of allowed login
     * attempts. Return Integer.MAX_VALUE to allow an infinite number of login
     * attempts.
     ********************************************************************************/

     protected int getMaxLoginAttemptsImpl() throws SmartCardException
     {
          return 5;
     }

     /********************************************************************************
     * Provide a method that returns the number of allowed login attempts
     * remaining (before the smart card will lock). Return Integer.MAX_VALUE if
     * the smart card will never lock.
     ********************************************************************************
     /
     protected int getRemainingLoginAttemptsImpl() throws SmartCardException
     {
          return 4;
     }

     /********************************************************************************
     * Provide a method that logs into the smart card with a specified password.
     * This method should not implement UI.
     ********************************************************************************/

     protected boolean loginImpl( String password ) throws SmartCardException
     {
        // Create the APDU command for your smart card. Consult documentation
        // from your smartcard vendor.
```

```
        CommandAPDU command =
         new CommandAPDU( (byte)0x00, (byte)0x20, (byte)0x00, (byte)0x00 );
        command.setLcData( password.getBytes() );
        ResponseAPDU response = new ResponseAPDU();
        sendAPDU( command, response );

      // Check for response codes specific to your smart card.
        if ( response.checkStatusWords( (byte)0x90, (byte)0x00 ) )
        {
            return true;
        }
        else if ( response.checkStatusWords( (byte)0x64, (byte)0xF8 ) )
        {
            throw new SmartCardLockedException();
        }
        else
        {
           // Authentication failed
            return false;
        }
    }

    /*************************************************************************
    * Provide a method that returns an ID for the smart card associated with
    * this session.
    *************************************************************************/
    protected SmartCardID getSmartCardIDImpl() throws SmartCardException
    {
        // Retrieve a unique ID from the card

        byte [] uniqueCardData = new byte[]
        { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b };

        // Convert the byte array to a long

        SHA1Digest digest = new SHA1Digest();
        digest.update( uniqueCardData );
        long idLong = byteArrayToLong(Arrays.copy(digest.getDigest(), 0, 8));

        return new SmartCardID( idLong , ID_STRING, getSmartCard() );
    }


    /*************************************************************************
    * Private method to convert a byte array into a long integer. Note that the
    * return value should be interpretted as an unsigned value.
    *
    * The method throws an IllegalArgumentException if the byte array contains
    * a number larger than 64 bits.
    *
    * Note: If your smart card driver is only designed to work with BlackBerry
    * Version 4.2 or later, you can replace this method with a call to
    * CryptoByteArrayArithmetic.valueOf( byte [] ).
    *************************************************************************/
```

```
    private long byteArrayToLong( byte[] array )
    {
        if ( array == null )
        {
            throw new IllegalArgumentException();
        }
        // Remove the leading zeros from the specified byte array and return
        // a new byte array without them.

        int zeros = 0;
        for ( int i = 0; i < array.length && array[i] == 0; i++ )
        {
            zeros++;
        }

        if ( zeros != 0 )
        {
            array = Arrays.copy( array, zeros, array.length - zeros );
        }
        int length = array.length;

        if( length > 8 )
        {
            throw new IllegalArgumentException();
        }
        long n = 0;

        for( int i=0; i<length; i++ )
        {
            n <<= 8;
            n += array[i] & 0xff;
        }
        return n;
    }


    /*****************************************************************************
    * Provide a method that returns random data from the smart cards internal
    * random number generator.
    *****************************************************************************/
    protected byte [] getRandomBytesImpl( int maxBytes ) throws SmartCardException
    {
        // Create the appropriate CommandAPDU for your smart card. See your
        // smart card vendor's documentation.

        CommandAPDU command =
         new CommandAPDU((byte)0x00,(byte)0x4C,(byte)0x00,(byte)0x00,maxBytes);
        ResponseAPDU response = new ResponseAPDU();
        sendAPDU( command, response );

        // Check for response codes specific to your smart card

        if( response.checkStatusWords( (byte)0x90, (byte)0x00 ) )
        {
        // return the response code containing the random data
            return response.getData();
```

```
          }
          return null;
     }

     /**************************************************************************
     * Provide a method that returns certificates from the smart card.
     * Return an array containing copies of certificates on the card.
     **************************************************************************/
     protected CryptoSmartCardKeyStoreData[] getKeyStoreDataArrayImpl()
      throws SmartCardException, CryptoTokenException
     {
          try
          {
               // Display a progress dialog because this operation may take a long
               // time.

               displayProgressDialog( WAITING_MSG, 4 );

               // Associate the certificates with a particular card.

               SmartCardID smartCardID = getSmartCardID();
               RSACryptoToken token = new MyRSACryptoToken();
               RSACryptoSystem cryptoSystem = new RSACryptoSystem( token, 1024 );
               RSAPrivateKey privateKey;
               CryptoSmartCardKeyStoreData[] keyStoreDataArray = new
                CryptoSmartCardKeyStoreData[ 3 ];

               // This encoding would be extracted from the card using a series of
               // APDU commands.

               Certificate certificate = null;

               // Extract the certificate encoding from the card.

               byte [] certificateEncoding = new byte[0];
               try
               {
                    certificate = new X509Certificate( certificateEncoding );
               }
               catch( CertificateParsingException e )
               {
                    // invalid X509 certificate
               }
               }

               stepProgressDialog( 1 );
               privateKey = new RSAPrivateKey(cryptoSystem,
                 new MyCryptoTokenData( smartCardID, ID_PKI ) );
               keyStoreDataArray[ 0 ] =
               new CryptoSmartCardKeyStoreData( null, ID_CERT, privateKey, null,
                 KeyStore.SECURITY_LEVEL_HIGH, certificate, null, null, 0 );

               stepProgressDialog( 1 );
               privateKey = new RSAPrivateKey( cryptoSystem,
                 new MyCryptoTokenData( smartCardID, SIGNING_PKI ) );
               keyStoreDataArray[ 1 ] =
```

```
                new CryptoSmartCardKeyStoreData( null, SIGNING_CERT, privateKey,
                  null, KeyStore.SECURITY_LEVEL_HIGH, certificate, null, null, 0 );

                stepProgressDialog( 1 );
                privateKey = new RSAPrivateKey( cryptoSystem,
                 new MyCryptoTokenData( smartCardID, ENCRYPTION_PKI ) );
                keyStoreDataArray[ 2 ] = new CryptoSmartCardKeyStoreData( null,
                  ENCRYPTION_CERT, privateKey, null,
                  KeyStore.SECURITY_LEVEL_HIGH, certificate, null, null, 0 );

                stepProgressDialog( 1 );

                // Sleep so the user sees the last step of the progress dialog
                // as it moves to 100%

                try
                {
                     Thread.sleep( 250 );
                }
                catch ( InterruptedException e )
                {
                }
                dismissProgressDialog();
                return keyStoreDataArray;
           }
           catch ( CryptoUnsupportedOperationException e )
           {
           }
           catch ( UnsupportedCryptoSystemException e )
           {
           }
           catch ( CryptoTokenException e )
           {
           }
           throw new SmartCardException();
      }

      /**************************************************************************
      * Send some data to the smart card for signing or decryption.
      **************************************************************************/
      /*package*/ void signDecrypt( RSACryptoSystem cryptoSystem,
        MyCryptoTokenData privateKeyData,byte[] input, int inputOffset,
        byte[] output, int outputOffset ) throws SmartCardException
      {
      // Check for nulls
      if ( cryptoSystem == null || privateKeyData == null || input == null || output
== null) {
      throw new IllegalArgumentException();
      }
      // Validate the input parameters
      int modulusLength = cryptoSystem.getModulusLength();
      if ( ( input.length < inputOffset + modulusLength ) || ( output.length <
outputOffset +
      modulusLength ) ) {
      throw new IllegalArgumentException();
      }
```

```
     // Construct the response Application Protocol Data Unit
     ResponseAPDU response = new ResponseAPDU();
     // Construct the command and set its information
     // Create a CommandAPDU which your smart card will understand
     CommandAPDU signAPDU = new CommandAPDU( (byte)0x80, (byte)0x56, (byte)0x00,
     (byte)0x00, modulusLength );
     signAPDU.setLcData( input, inputOffset, input.length - inputOffset );
     // Send the command to the smart card
     sendAPDU( signAPDU, response );
     // Validate the status words of the response
     // Check for response codes specific to your smart card
     if ( response.checkStatusWords( (byte)0x90, (byte)0x00 ) ) {
     byte [] responseData = response.getData();
     System.arraycopy( responseData, 0, output, outputOffset, responseData.length );
     }
     else {
     throw new SmartCardException( “Invalid response code, sw1=” +
Integer.toHexString(
     response.getSW1() & 0xff ) + “ sw2=” + Integer.toHexString( response.getSW2() &
0xff ) );
     }
     }
}
```

Code sample: Implementing a class to represent a type of smart card

```
package com.rim.samples.device.smartcard;
import net.rim.device.api.smartcard.*;
import net.rim.device.api.util.*;
import net.rim.device.api.crypto.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.system.*;

/**************************************************************************
* This class represents a kind (or model or family) of physical smart card.
* There should only be one instance of this class in the system at a time. The
* instance is managed by the SmartCardFactory.
**************************************************************************/

public class MyCryptoSmartCard extends CryptoSmartCard implements Persistable
{
     private final static byte MY_ATR [] = {
     (byte)0x3b, (byte)0x7d, (byte)0x11,
     (byte)0x00, (byte)0x00, (byte)0x00,
     (byte)0x31, (byte)0x80, (byte)0x71,
     (byte)0x8e, (byte)0x64, (byte)0x86,
     (byte)0xd6, (byte)0x01, (byte)0x00,
     (byte)0x81, (byte)0x90, (byte)0x00 };

     private final static AnswerToReset _myATR    = new AnswerToReset( MY_ATR );
     private static final String LABEL            = "RIM Sample";
```

```
    private static final String DISPLAY_SETTINGS = "Show driver settings now";
    private static final String RSA              = "RSA";

    /*************************************************************************
    * The libMain() method is invoked when the BlackBerry device starts and
    * registers this smart card driver with the smart card factory.
    *
    * Registering this smart card driver with the smart card factory
    * automatically registers the driver with the user authenticator framework
    * which allows the smart card to be used as a second factor of
    * authentication for unlocking a BlackBerry device.
    *
    * In the BlackBerry development environment, in the settings for the smart
    * card driver project, set the project type to Library and select the
    * auto-run on startup' setting.
    *************************************************************************/

    public static void libMain( String args[] )
    {
         try
         {
              SmartCardFactory.addSmartCard( new MyCryptoSmartCard() );
         }
         catch(ControlledAccessException cae)
         {
          // Application control may not allow your driver to be used with the
          // user authenticator framework, in which case it will throw a
          // ControlledAccessException.
          // Your driver is registered with the smart card API framework and
          //can still
          // be used for importing certificates and signing/decrypting messages.
         }
    }

    /*********************************************************
    * Retrieve the session handler for this smart card.
    * Implementations of this method should not include any UI.
    *********************************************************/
    protected SmartCardSession openSessionImpl( SmartCardReaderSession
readerSession )
     throws SmartCardException
    {
         return new MyCryptoSmartCardSession( this, readerSession );
    }

    /*************************************************************************
    * Determine if the system should use this smart card object
    * to communicate with a physical smart card that has the given AnswerToReset.
    * The system invokes this method to determine which smart card driver
    * implementation it should use to communicate with a physical smart card
    * found in a BlackBerry Smart Card Reader.
    *************************************************************************/
    protected boolean checkAnswerToResetImpl( AnswerToReset atr )
    {
       /*********************************************************************
       * If this method returns false, the cryptographic smart card driver will
```

```
         * not be used to perform additional operations on a particular smart card.
         * This method should only return true if you support the specified ATR.
         *
         * If this method returns true, but there is no support for the smart card
         * that corresponds to the ATR,other smart card drivers may stop working
         * properly.
         *
         * The AnswerToReset argument contains the full ATR from the smart card.
         * Your implementation of this method may check the entire ATR or just
         * parts of the ATR, as long as the driver supports the corresponding
         * smart card.
         ***************************************************************************/

           return _myATR.equals( atr );
      }

      /***************************************************************************
      * Retrieve the label associated with this smart card.
      * The string you return should not include the words "smart card", as the
      * system uses this method to generate strings such as
      * "Please insert your 'label' smart card".
      ***************************************************************************/
      protected String getLabelImpl()
      {
          return LABEL;
      }

      /*****************************************
      * Retrieves this smart card's capabilities
      *****************************************/
      protected SmartCardCapabilities getCapabilitiesImpl()
      {
           return new SmartCardCapabilities( SmartCardCapabilities.PROTOCOL_T0 );
      }

      /*********************************************************
      * Indicate whether this smart card can display its settings.
      *********************************************************/
      protected boolean isDisplaySettingsAvailableImpl( Object context )
      {
           return true;
      }

      /***************************************************************************
      * Display this smart card's settings.
      * This method will be invoked from the smart card options screen when
      * the user selects the driver and chooses to view the settings of that
      * driver.
      *
      * This method could be called from the event thread. The driver should not
      * block the event thread for long periods of time.
      *
      * The context parameter is reserved for possible future use.
      ***************************************************************************/
      protected void displaySettingsImpl( Object context )
      {
```

```
            Dialog.alert( DISPLAY_SETTINGS );
        }

        /**********************************************************
        * Retrieve the algorithms supported by this smart card.
        * You must return one or more of “RSA”, “DSA”, or “ECC”.
        **********************************************************/
        public String[] getAlgorithms()
        {
            return new String [] { "RSA" };
        }

        /**********************************************************************
        * Retrieve a crypto token that supports the given algorithm.
        * The argument algorithm is the name of the algorithm.
        * Return the Crypto token supporting the named algorithm.
        *
        * Throw NoSuchAlgorithmException if the specified algorithm is invalid.
        * Throw CryptoTokenException if there is a token-related problem.
         **********************************************************************/
       public CryptoToken getCryptoToken( String algorithm )
         throws NoSuchAlgorithmException, CryptoTokenException
        {
            if ( algorithm.equals( RSA ) )
            {
                return new MyRSACryptoToken();
            }
            throw new NoSuchAlgorithmException();
        }
}
```

7 Display cryptographic smart card driver options

If your implementation of `isDisplaySettingsAvailableImpl()` returns true.

1. On the BlackBerry device, click **Options > Security Options > Smart Card**.
2. In the **Registered Card Drivers** section select the cryptographic smart card driver you are testing.
3. Click **Driver Settings**.

Testing smart card drivers

8

Setting up a BlackBerry device to test a cryptographic smart card driver

Set up the BlackBerry Smartphone Simulator to test a cryptographic smart card driver

To test a cryptographic smart card driver with the BlackBerry Smartphone Simulator, you require the Casira End Point. You do not require the Casira Bluetooth hardware and software development system. For more information, visit Cambridge Silicon Radio Ltd. at www.btdesigner.com/devcasira.htm.

1. On the taskbar, click **Start > Programs > Research In Motion > BlackBerry JDE 5.0.0 > JDE**.
2. On the main menu, click **Edit > Preferences**.
3. Click the **Simulator** tab.
4. Click the **Ports** tab.
5. In the **Bluetooth test board port** field, type the port information.
6. Click **OK**.

Test signing and decrypting of S/MIME email messages

1. On the BlackBerry device, click **Options > Security Options > Certificates**.
2. Click **Import Smart Card Certs**.
3. If required, type your PIN.
4. Select the certificate you want to import from the smart card.
5. Type the smart card password.
6. Specify S/MIME to sign and decrypt email messages.

Set up to test signing and decrypting S/MIME email messages

The following steps require a BlackBerry Smart Card Reader.

1. Install the S/MIME Support Package for BlackBerry smartphones. See the *Security Technical Overview - S/MIME Support Package for BlackBerry smartphones* for more information about installing the S/MIME Support package for BlackBerry smartphones on your computer or a BlackBerry device.
2. Connect the BlackBerry device to the computer.
3. At a command prompt, switch to the BlackBerry Java Development Environment bin folder.

4. Type the following command:

```
javaloader [-usb] [-wpassword] load <file>
```

- *password*:If a password is set, the password for the BlackBerry device
- *file*:The .cod file that the cryptographic smart card driver downloads to the BlackBerry device

Test two-factor authentication

1. On the BlackBerry device, click **Options > Security Options > General Settings**.
2. Verify that your smart card is inserted in the BlackBerry Smart Card Reader.
3. Change the **User Authenticator Password** field to **Enabled**.
4. Click **Save**.
5. When prompted, type the smart card password.
6. Click **Enter**.

Advanced Security SD card support

9

A BlackBerry device with BlackBerry Device Software version 5.0 or later supports the use of Advanced Security SD (ASSD) cards. ASSD cards are flash memory cards that incorporate smart card functionality. They are defined by the *Advanced Security SD Extension Specification Version 2.00 (April 16, 2009)* published by the SD Card Association. Version 1.10 of that document was titled *Mc-EX Extension Specification*.

Draft version 2.00 of the specification defines a Security System as a set of card commands that perform security-related operations. The specification allows for various Security Systems which are identified by integer indexes beginning with 0 and ending with 15. BlackBerry devices only support the Mc-EX Security System, which is assigned index 0. The Mc-EX Security System is owned by the 5C Group and is based upon ISO-7816.

ASSD cards do not return ATR sequences. Instead, the reader driver for ASSD cards uses data from the CID register of the card to construct an ATR sequence. The smart card infrastructure uses the ATR that the driver constructs to support the ASSD card as a smart card.

Data formats for virtual ATR sequences

Advanced Security SD cards cannot return an ATR sequence. To use an Advanced Security SD card as a smart card with a BlackBerry device, the reader driver constructs a virtual ATR sequence that is based on the CID register of the Advanced Security SD card. The following tables describe the formats for a virtual ATR sequence. Interface characters are not specified in the virtual ATR sequence.

Bit range	ISO-7816 Name	Value	Width
135-128	TS	0x3B	8
127-120	T0	0x0F	8
119-8	T[1..14]	from the media card CID register	112
7-0	TCK	calculated to make XOR of {T0-T14,TCK} equal zero	8

Historical bytes format T[1..14]

Bit range	ISO-7816 Characters	Value	Width
119-112	T1	0xFF	8
111-104	T2	value of MID CID register field	8
103-88	T[3..4]	value of OID CID register field	16
87-48	T[5..9]	value of PNM CID register field	40
47-40	T10	value of PRV CID register field	8
39-8	T[11..14]	value of PSN CID register field	32

Glossary

10

Advanced Security SD card

An Advanced Security SD card is a media card that complies with the Advanced Security SD Extension Specification that the SD Association developed. BlackBerry devices support only microSD cards that use the MCEX security system.

API

application programming interface

CAC

Common Access Card

CID

Card Identification

JVM

Java Virtual Machine

JSR

Java Specification Request

PIV

Personal Identity Verification

Provide feedback

11

To provide feedback on this deliverable, visit www.blackberry.com/docsfeedback.

Document revision history

12

Date	Description
6 April 2010	Added the following topics: • Smart card drivers • Smart card driver infrastructure • ASSD card support • Virtual ATR data format Updated the following topics: • CryptoSmartCard methods you implement to create a driver • CryptoSmartCardSession methods you implement to create a driver

Legal notice

13

REFERENCED HEREIN INCLUDING WITHOUT LIMITATION ANY OF THE FOLLOWING DAMAGES: DIRECT, CONSEQUENTIAL, EXEMPLARY, INCIDENTAL, INDIRECT, SPECIAL, PUNITIVE, OR AGGRAVATED DAMAGES, DAMAGES FOR LOSS OF PROFITS OR REVENUES, FAILURE TO REALIZE ANY EXPECTED SAVINGS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, LOSS OF BUSINESS OPPORTUNITY, OR CORRUPTION OR LOSS OF DATA, FAILURES TO TRANSMIT OR RECEIVE ANY DATA, PROBLEMS ASSOCIATED WITH ANY APPLICATIONS USED IN CONJUNCTION WITH RIM PRODUCTS OR SERVICES, DOWNTIME COSTS, LOSS OF THE USE OF RIM PRODUCTS OR SERVICES OR ANY PORTION THEREOF OR OF ANY AIRTIME SERVICES, COST OF SUBSTITUTE GOODS, COSTS OF COVER, FACILITIES OR SERVICES, COST OF CAPITAL, OR OTHER SIMILAR PECUNIARY LOSSES, WHETHER OR NOT SUCH DAMAGES WERE FORESEEN OR UNFORESEEN, AND EVEN IF RIM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW IN YOUR JURISDICTION, RIM SHALL HAVE NO OTHER OBLIGATION, DUTY, OR LIABILITY WHATSOEVER IN CONTRACT, TORT, OR OTHERWISE TO YOU INCLUDING ANY LIABILITY FOR NEGLIGENCE OR STRICT LIABILITY.

THE LIMITATIONS, EXCLUSIONS, AND DISCLAIMERS HEREIN SHALL APPLY: (A) IRRESPECTIVE OF THE NATURE OF THE CAUSE OF ACTION, DEMAND, OR ACTION BY YOU INCLUDING BUT NOT LIMITED TO BREACH OF CONTRACT, NEGLIGENCE, TORT, STRICT LIABILITY OR ANY OTHER LEGAL THEORY AND SHALL SURVIVE A FUNDAMENTAL BREACH OR BREACHES OR THE FAILURE OF THE ESSENTIAL PURPOSE OF THIS AGREEMENT OR OF ANY REMEDY CONTAINED HEREIN; AND (B) TO RIM AND ITS AFFILIATED COMPANIES, THEIR SUCCESSORS, ASSIGNS, AGENTS, SUPPLIERS (INCLUDING AIRTIME SERVICE PROVIDERS), AUTHORIZED RIM DISTRIBUTORS (ALSO INCLUDING AIRTIME SERVICE PROVIDERS) AND THEIR RESPECTIVE DIRECTORS, EMPLOYEES, AND INDEPENDENT CONTRACTORS.

IN ADDITION TO THE LIMITATIONS AND EXCLUSIONS SET OUT ABOVE, IN NO EVENT SHALL ANY DIRECTOR, EMPLOYEE, AGENT, DISTRIBUTOR, SUPPLIER, INDEPENDENT CONTRACTOR OF RIM OR ANY AFFILIATES OF RIM HAVE ANY LIABILITY ARISING FROM OR RELATED TO THE DOCUMENTATION.

Prior to subscribing for, installing, or using any Third Party Products and Services, it is your responsibility to ensure that your airtime service provider has agreed to support all of their features. Some airtime service providers might not offer Internet browsing functionality with a subscription to the BlackBerry® Internet Service. Check with your service provider for availability, roaming arrangements, service plans and features. Installation or use of Third Party Products and Services with RIM's products and services may require one or more patent, trademark, copyright, or other licenses in order to avoid infringement or violation of third party rights. You are solely responsible for determining whether to use Third Party Products and Services and if any third party licenses are required to do so. If required you are responsible for acquiring them. You should not install or use Third Party Products and Services until all necessary licenses have been acquired. Any Third Party Products and Services that are provided with RIM's products and services are provided as a convenience to you and are provided "AS IS" with no express or implied conditions, endorsements, guarantees, representations, or warranties of any kind by RIM and RIM assumes no liability whatsoever, in relation thereto. Your use of Third Party Products and Services shall be governed by and subject to you agreeing to the terms of separate licenses and other agreements applicable thereto with third parties, except to the extent expressly covered by a license or other agreement with RIM.

Certain features outlined in this documentation require a minimum version of BlackBerry® Enterprise Server, BlackBerry® Desktop Software, and/or BlackBerry® Device Software.

Research In Motion Limited
295 Phillip Street
Waterloo, ON N2L 3W8
Canada

Research In Motion UK Limited
Centrum House
36 Station Road
Egham, Surrey TW20 9LF
United Kingdom

Published in Canada